CS304 Practice Final

Worth: 30% (or 100% if you score better than your combined average on assignments + midterm)

Instructions: 3 hours, pen and paper only. This practice exam is slightly longer than the real final.

Q1: give the full set notation definition of Θ notation. It begins like "*we define by* $\Theta(n)$ *the set of all functions …*" and draw the accompanying plot showing $f(n)$ and $g(n)$.

Q2: do the same as Q1 but for O notation.

Q3: show that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$ by determining positive constants c_1, c_2 and n_0 according to your definition from Q1.

Q4: write the following function in O notation: $6 \cdot \log(n) \cdot n + 36 \cdot n - 3n^2$

Q5: what is the time complexity in O notation of the following pseudocode?

```
binarySearch(arr, x, low, high)
    repeat till low = high
           mid = (low + high)/2
               if (x == arr[mid])
               return mid

               else if (x > arr[mid]) // x is on the right side
                   low = mid + 1

               else                  // x is on the left side
                   high = mid - 1
```

Q6: write a c++ function with $O(n + nlogn)$ time complexity (skeleton below). The function doesn't need to do anything useful, it just needs to run in $O(n + logn)$ time.

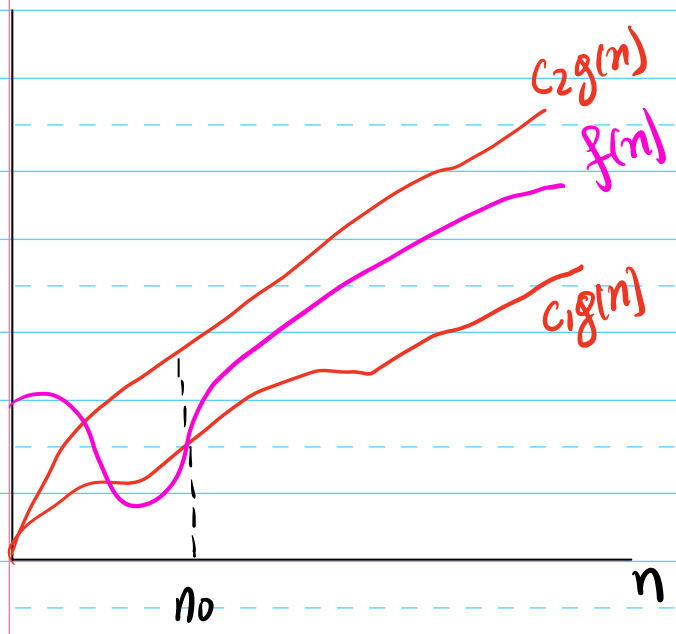
```
void fun(int n){…}
```

Q7: below is the code for insertion sort with an accompanying main. Show what this program will print when it runs.

```
template <typename T>
void insertion_sort(std::vector<T>& v) {
    for (int j = 1; j < v.size(); j++) {
        auto key = std::move(v[j]);
        int i = j - 1;
        while (i > -1 && key < v[i]) {
            v[i + 1] = std::move(v[i]);
            i--;
        }
        v[i + 1] = std::move(key);
        std::cout << v[0] << std::endl;
    }
}

int main()
{
    std::vector<int> input = { 4,2,5,1 };
    insertion_sort(input);
}
```

Answer ① theta notation.

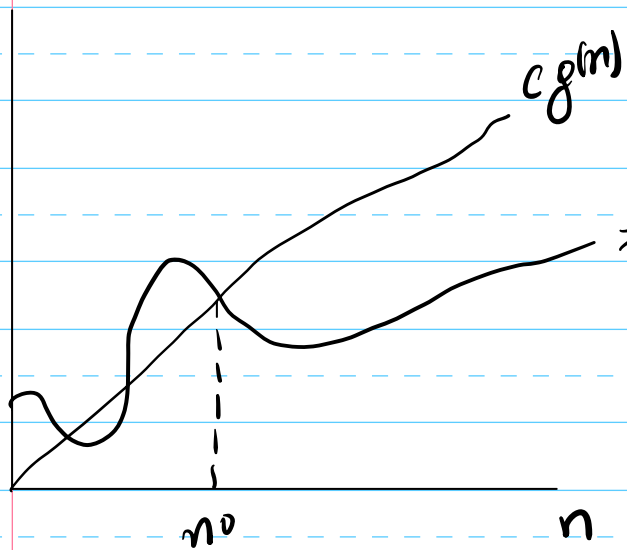
$\Theta(g(n)) = \{f(n)$: there exist positive Constants c_1, c_2, and n_0 Such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$



$f(n) = \Theta(g(n))$

answer ② O notation

$O(g(n)) = \{f(n)$: there exist positive Constants c and n_0 Such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$



$f(n) = O(g(n))$

Answer 3 Show that $2n+3 = O(n)$ by determining positive constants C_1, C_2 and n_0

$$1 \times n \leq 2n+3 \leq 2n+3n$$

$$1n \leq 2n+3 \leq 5n \qquad n \geq 1$$

$1n$ → C_1, $g(n)$; $5n$ → C_2, $g(n)$

$C_1 = 1$, $C_2 = 5$ and $n_0 = 1$

Example 2 $2n^2+3n+4$

$$1n^2 \leq 2n^2+3n+4 \leq 2n^2+3n^2+4n^2$$

$$1 \times n^2 \leq 2n^2+3n+4 \leq 9n^2 \qquad n \geq 1$$

$1n^2$ → C_1, $g(n)$; $9n^2$ → C_2, $g(n)$

$n_0 = 1$

Answer 4 Write the following in O notation

$6n\log(n) + 36n - 3n^2$

O notation is n^2 , term with the highest exponent

Answer 5 time Complexity is O(logn)

```
binarySearch(arr, x, low, high)
    repeat till low = high
            mid = (low + high)/2
                if (x == arr[mid])
                return mid

                else if (x > arr[mid]) // x is on the right side
                    low = mid + 1

                else                    // x is on the left side
                    high = mid - 1
```

narrow the array by half.

recursive version is O(logn)

Answer 6

Q6: write a c++ function with $O(n + nlogn)$ time complexity (skeleton below). The function doesn't need to do anything useful, it just needs to run in $O(n + logn)$ time.

```
void MyFunction (int n) {
    // O(n) operations
  for (int i=0; i<n; i++) {
      // do something
    }
  // O(n log n) operations
  for (int i=1; i<n; i*=2){
      // do something
    }
  }
int main () {
  int n=100;
  MyFunction(n);
  return 0;
}
```

Question 7

Q7: below is the code for insertion sort with an accompanying main. Show what this program will print when it runs.

```
template <typename T>
void insertion_sort(std::vector<T>& v) {
    for (int j = 1; j < v.size(); j++) {
        auto key = std::move(v[j]);
        int i = j - 1;
        while (i > -1 && key < v[i]) {
            v[i + 1] = std::move(v[i]);
            i--;
        }
        v[i + 1] = std::move(key);
        std::cout << v[0] << std::endl;
    }
}

int main()
{
    std::vector<int> input = { 4,2,5,1 };
    insertion_sort(input);
}
```

output : 2, 2, 1

Q8: below is the code for quick sort. Modify this code so it runs in $O(nlogn)$ time on sorted input.

```
int partition(std::vector<int>& arr, int p, int r)
{
	int pivot = arr[r];
	int i = p - 1;
	for (int j = p; j < r; ++j)
		if (arr[j] <= pivot)
		{
			++i;
			std::swap(arr[i], arr[j]);
		}
	std::swap(arr[i + 1], arr[r]);

	return i + 1;
}

void quicksort(std::vector<int>& arr, int p, int r)
{
	if (p < r)
	{
		int q = (p + r) / 2;
		quicksort(arr, p, q - 1);
		quicksort(arr, q + 1, r);
	}
}
```

Q9: write the copy constructor and move assignment operator for the following class:

```
template <typename T>
class LargeTypeRaw {
public:
    // Default Constructor
    explicit LargeTypeRaw(int size = 10)
        : size{ size }, arr{ new T[size] }
    {}

    // Destructor
    ~LargeTypeRaw() {
        delete[] arr;
    }
    bool operator<(const LargeTypeRaw& rhs) {
        return (size < rhs.get_size());
    }

    int get_size() const {
        return size;
    }

private:
    int size;
    T* arr;
};
```

Q10: fill in the time complexity in O notation for the following sorting algorithms on the following input types:

Algorithm/input type	Sorted array	Random array	Array of all zeros
Insertion sort			
Selection sort			
Mergesort			
Quicksort			
Radix sort			
Treesort (using BST)			
Heapsort			

Answer 8

```
int partition(std::vector<int>& arr, int p, int r)
{
        int pivot = arr[r];
        int i = p - 1;
        for (int j = p; j < r; ++j)
                if (arr[j] <= pivot)
                {
                        ++i;
                        std::swap(arr[i], arr[j]);
                }
        std::swap(arr[i + 1], arr[r]);

        return i + 1;
}

void quicksort(std::vector<int>& arr, int p, int r)
{
        if (p < r)
        {
```

// check if the array is already sorted.

if (r-p ≤10) { // Arbitrary threshold for small array

std::sort (arr.begin() + p, arr.begin()+r+1);

return;

}

```
                int q = (p + r) / 2;
                quicksort(arr, p, q - 1);
                quicksort(arr, q + 1, r);
        }
}
```

Answer 9

```
// move Constructor
LargetypeRaw(LargetypeRaw&& other) noexcept
    : size{ ther.size }, arr{other.arr}
{
    other.arr = nullptr;
}

// Copy assignment operator
LargetypeRaw& operator=(Const LargetypeRaw&
    other) {
    If (&other != this) {
        delete[] arr;
        size = other.size;
        array = new T[size];
    std::copy(other.arr, other.arr+size, arr);
    }
    return *this;
}
```

Answer 10

Q10: fill in the time complexity in O notation for the following sorting algorithms on the following input types:

Algorithm/input type	Sorted array	Random array	Array of all zeros
Insertion sort	$O(n)$	$O(n^2)$	$O(n)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Mergesort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quicksort	$O(n^2)$	$O(n \log n)$	$O(n^2)$
Radix sort	$O(d(n+k))$	$O(d(n+k))$	$O(d(n+k))$
Treesort (using BST)	$O(n^2)$	$O(n \log n)$	$O(n)$ or $O(n^2)$
Heapsort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q11: write c++ code to insert an element into the front of a singly-linked list (definition below)

```
//Definition for singly-linked list.
struct ListNode
{
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) { }
    ListNode(int x) : val(x), next(nullptr) { }
    ListNode(int x, ListNode* next) : val(x), next(next) { }
};
```

Q12: write c++ code to remove an element from a doubly-linked list (skeleton below, position is given)

```
void remove(Node* position){…}
```

Q13: insert the following elements into a binary search tree. Show the tree after each insertion.

```
Elems = [1,2,3,4,10,9,8,7,-1,2.5]
```

Q14: insert the following elements into a binary min heap. Show the heap after each insertion.

```
Elems = [5,4,0,2,1,6,3]
```

Q15: remove the following elements from the final tree in Q13, show the tree after each deletion.

```
Elems = [3,1,10]
```

Q16: show the heap from Q14 after 3 calls to deleteMin (show the heap after each call)

Q17: repeat Q13 but for an AVL tree.

Q18: remove the following elements from the final AVL tree in Q17, show the tree after each removal.

```
Elems = [1,2,3,4]
```

Q19: convert the following infix expression to postfix, and then evaluate the postfix expression

```
Expr = 3*(1+2)-(5+2)*7
```

Q20: convert the following infix expression into an expression tree, and then produce the prefix and postfix versions of the expression by using the preorder and postorder traversals

```
Expr = (1+2)*(3+4)*(5+6)-7
```

Q21: using the following hash function, insert the sequence A=[60,21,11,70,82] into a hash table with initial capacity 10. Use linear probing as the collision resolution method.

```
Hash function = key % TableSize
```

Q22: which operation is faster on average in a heap: `insert` or `deleteMin`?

Q23: write the C++ `rotateWithLeftChild(AvlNode*& k1)` code for an AVL tree

Q24: write the C++ `remove(const T& x, BinaryNode*& t)` for a BST

Answer 11

Q11: write c++ code to insert an element into the front of a singly-linked list (definition below)

```
//Definition for singly-linked list.
struct ListNode
{
    int val;
    ListNode* next;
    ListNode() : val(0), next(nullptr) { }
    ListNode(int x) : val(x), next(nullptr) { }
    ListNode(int x, ListNode* next) : val(x), next(next) { }
};
```

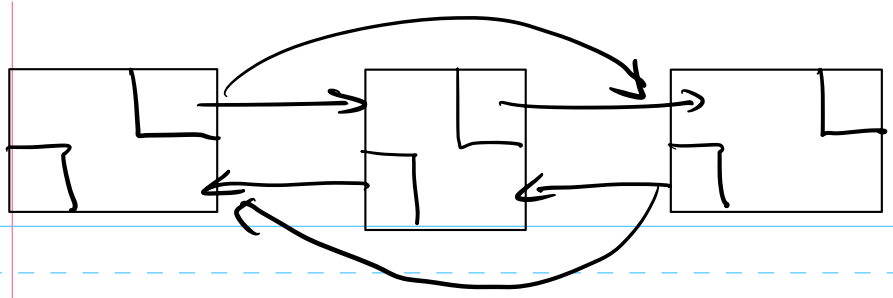
```
void insertionFront (listnode* &head, int val) {
  listnode* newNode = new listnode (val);
  newNode->next = head;
    head = newNode;
}
```

Answer 12

Q12: write c++ code to remove an element from a doubly-linked list (skeleton below, position is given)

```
void remove(Node* position){…}
```

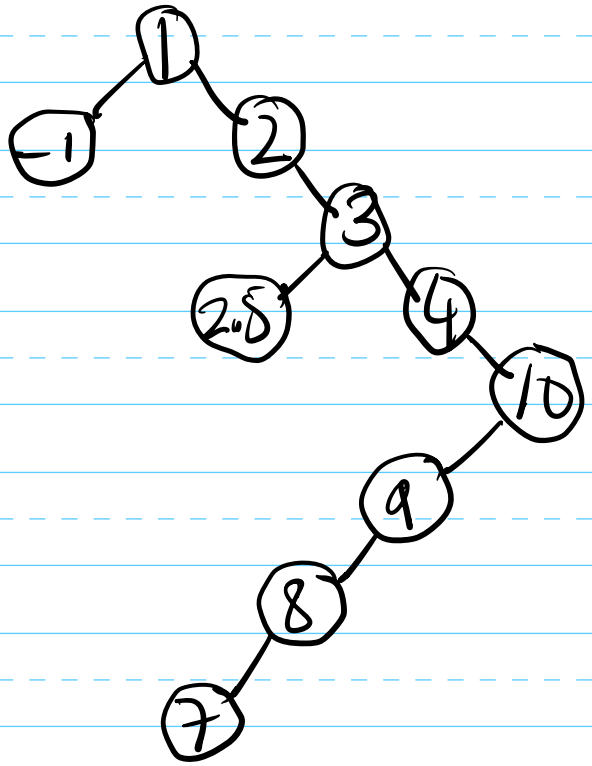
```
P->Prev->next = P->next;
P->Next->Prev = P->Prev;
```



Answer 13

Q13: insert the following elements into a binary search tree. Show the tree after each insertion.

```
Elems = [1,2,3,4,10,9,8,7,-1,2.5]
```



Answer 15

Q15: remove the following elements from the final tree in Q13, show the tree after each deletion.

```
Elems = [3,1,10]
```

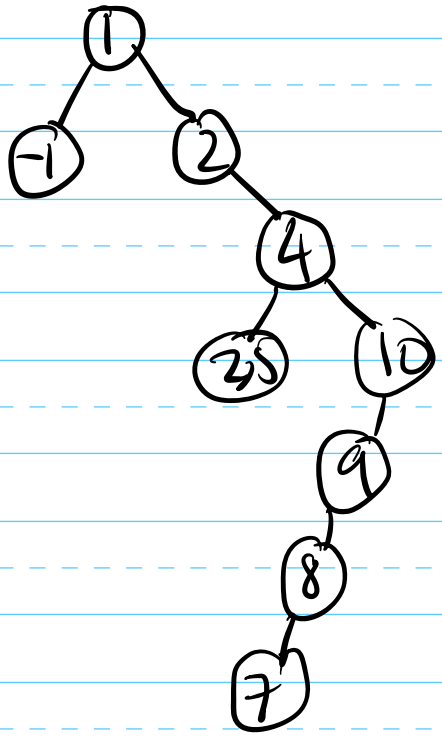
Deletion (3)

1. 0 child
2. 1 child
3. 2 children (Inorder Successor, Predecessor)

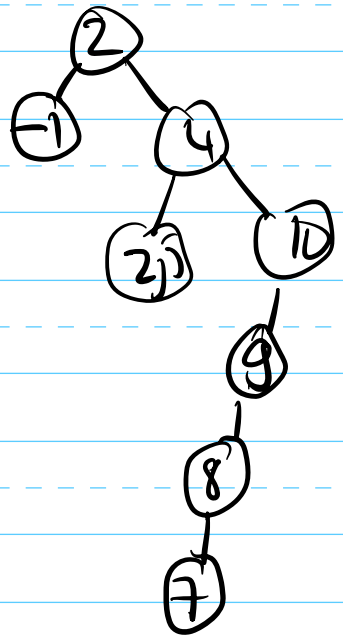
replace the # with the smallest element from the right subtree

replace the # with the largest element from the left subtree

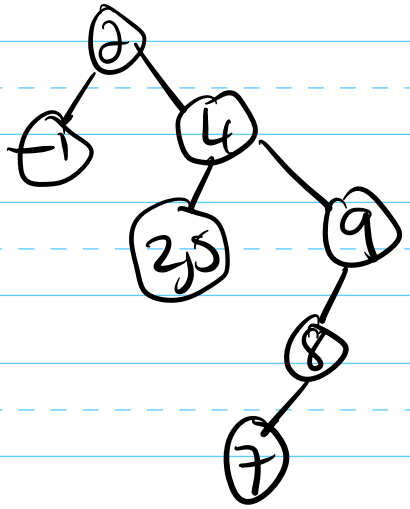
remove (3)



remove 11



remove 10



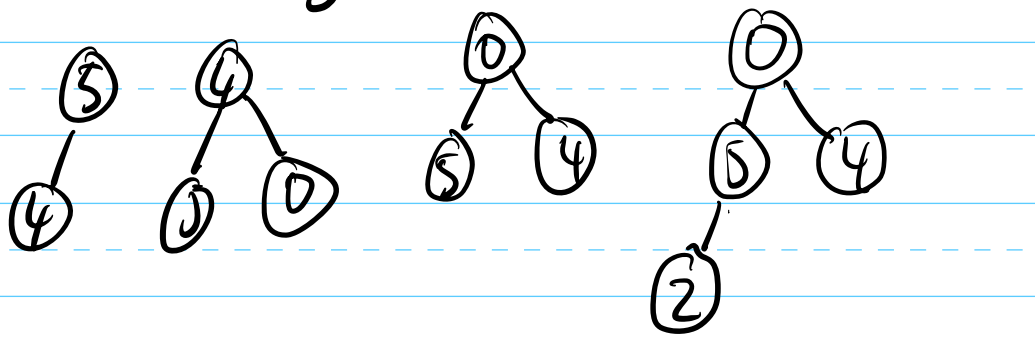
Answer (14)

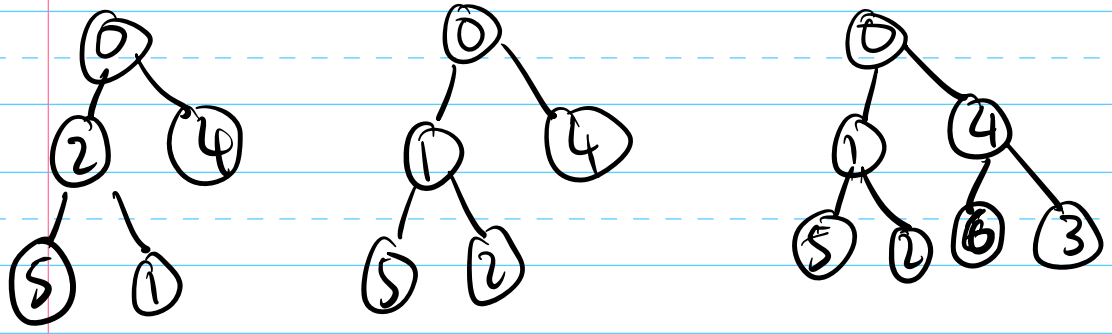
complete BST

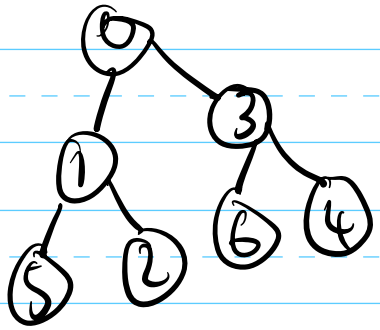
Q14: insert the following elements into a binary min heap. Show the heap after each insertion.

```
Elems = [5,4,0,2,1,6,3]
```

Min heap Parent node is less than or equal its child (except for root node (have any Parent))



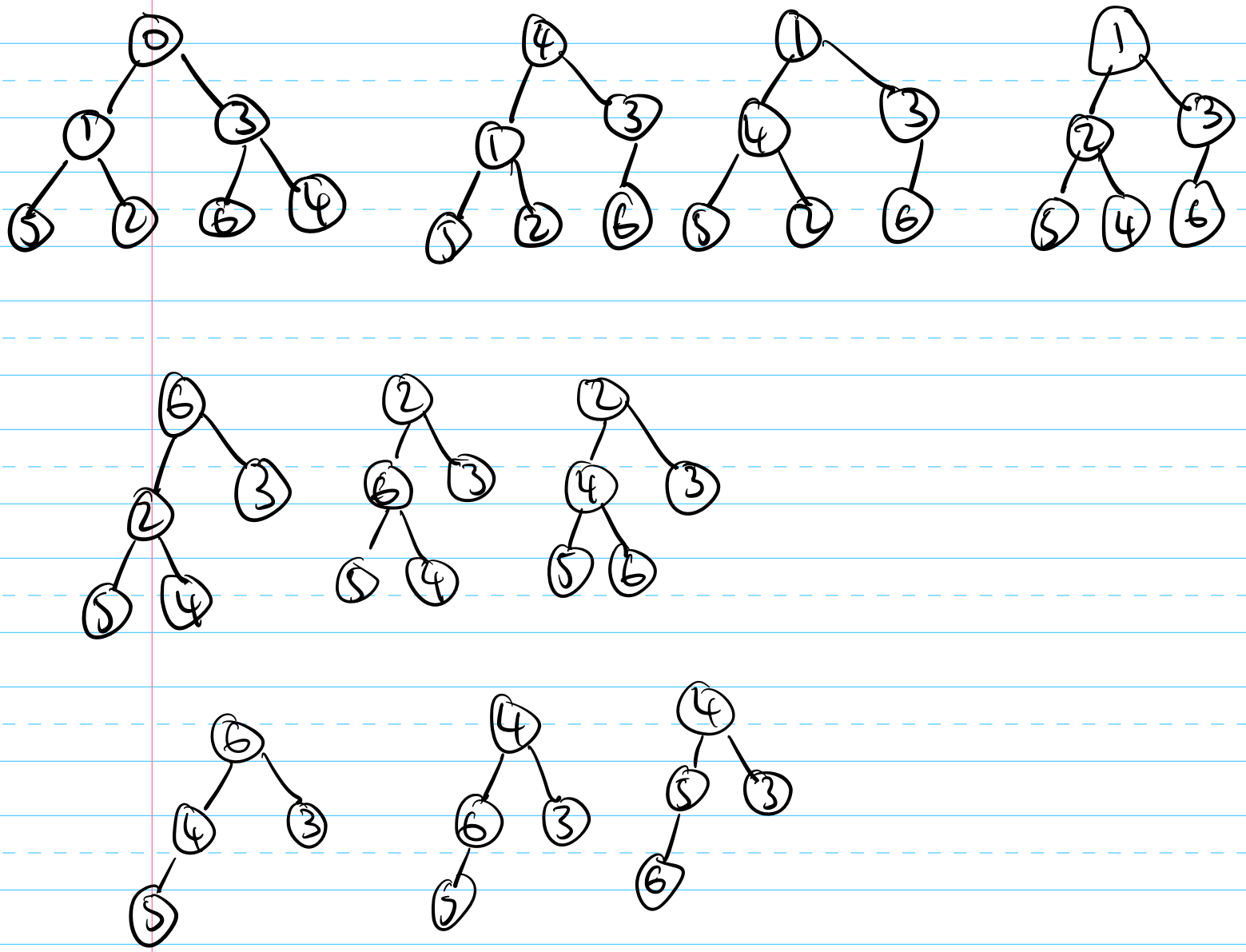




Answer 16

Q16: show the heap from Q14 after 3 calls to deleteMin (show the heap after each call)

elem = [5, 4, 0, 2, 1, 6, 3]



Answer 17

Q17: repeat Q13 but for an AVL tree.

```
Elems = [1,2,3,4,10,9,8,7,-1,2.5]
```

See assignment ④

Answer 18

See assignment ④

Q19: convert the following infix expression to postfix, and then evaluate the postfix expression

```
Expr = 3*(1+2)-(5+2)*7
```

stack =

output = 3 12+ * 5 2+ 7* −

Evaluate:

output: 9-49 = −40

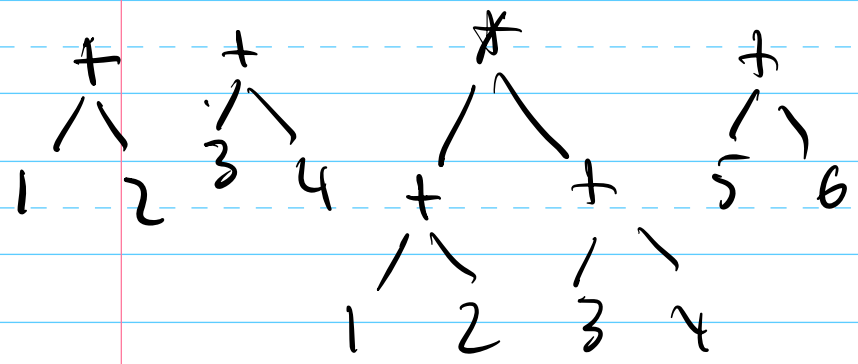
Answer 20

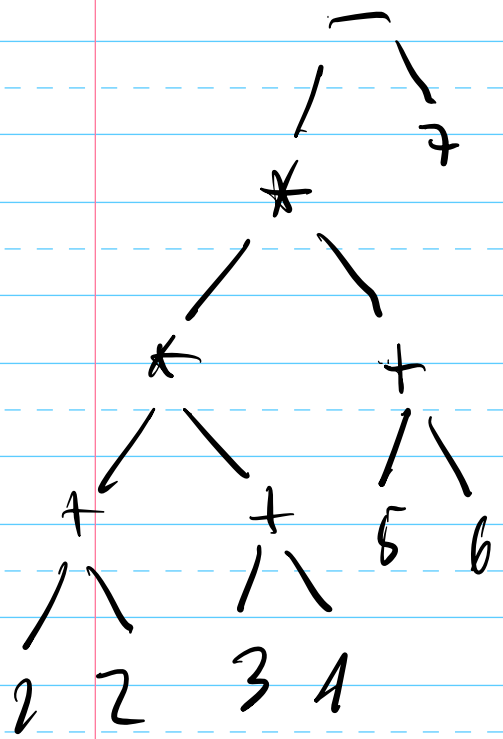
Q20: convert the following infix expression into an expression tree, and then produce the prefix and postfix versions of the expression by using the preorder and postorder traversals

```
Expr = (1+2)*(3+4)*(5+6)-7
```

Infix to Postfix

output = 12+34+*56+*7-





Postorder(node)

Preorder(node)

Q22: which operation is faster on average in a heap: `insert` or `deleteMin`?

Q23: write the C++ `rotateWithLeftChild(AvlNode*& k1)` code for an AVL tree

Q24: write the C++ `remove(const T& x, BinaryNode*& t)` for a BST

Why do we need a reference to a pointer?

A reference to a pointer allows us to indirectly access and modify the pointer variable itself

Answer 23

```
void rotateWithLeftChild (AvlNode* & k2)
    AvlNode* k1 = k2->left;
     k2->left = k1->right;
      k1->right = k2;
k2->height = max(height(k2->left), height(k2->right)) + 1;
k1->height = max(height(k1->left), k2->height) + 1;
          k2 = k1;
}
```

Answer 24

```
void remove(const T& x, BinaryNode*& t) {
    if(!t) return;
     if (x < t->val) remove (x, t->left);
      else if (t->val < x) remove (x, t->right);
      else if (t->left && t->right) {
          t->val = findmin(t->right)->val;
           remove (t->val, t->right);
       } else {
        BinaryNode* oldnode = t;
         t = (t->left) ? t->left : t->right;
           delete oldnode;
       }
}
```

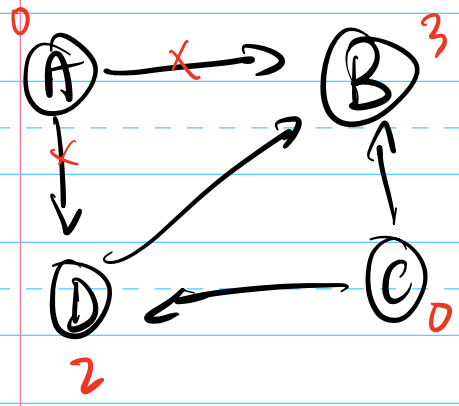
LeetCode

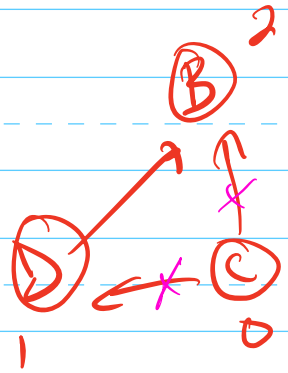
Search

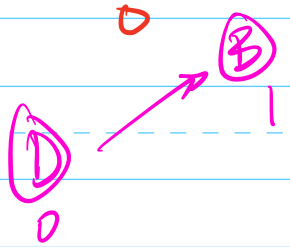
Red-black-tree.cpp | Hash-table.cpp | Two sums Untitled-1 ●

```cpp

class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        for(int i=0; i<nums.size(); i++){
            for(int j=i+1; j<nums.size(); j++){
                if(nums[i]+nums[j] == target)return{i,j};
            }
        }
        return {};
    }
};

Remove Duplicates from Sorted Array

class Solution {
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        for(int i=1; i<nums.size(); i++) {
            if(nums[i] == nums[i-1]) {
                nums[i-1] = INT_MAX;
            }
        }
        sort(nums.begin(), nums.end());
        int count = 0;
        for(int i=0; i<nums.size(); i++) {
            if(nums[i] != INT_MAX) count++;
        }

        //step 3 - return count
        return count;
    }
};

Remove Duplicates from Sorted list

class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* temp=head;
        while (temp&&temp->next)
        {
            if (temp->next->val==temp->val)
            {
                temp->next=temp->next->next;
                continue;
            }
            temp=temp->next;
        }
        return head;
    }
};
```

Topological ordering of a graph







Case I

A C D B